Software-defined and Programmable CPS/IoT OS: Architecting the Next Generation of CPS/IoT Operating Systems

Aniruddha Gokhale, Yogesh Barve, Anirban Bhattacharjee and Shweta Khare

a.gokhale@vanderbilt.edu

http://www.dre.vanderbilt.edu/~gokhale

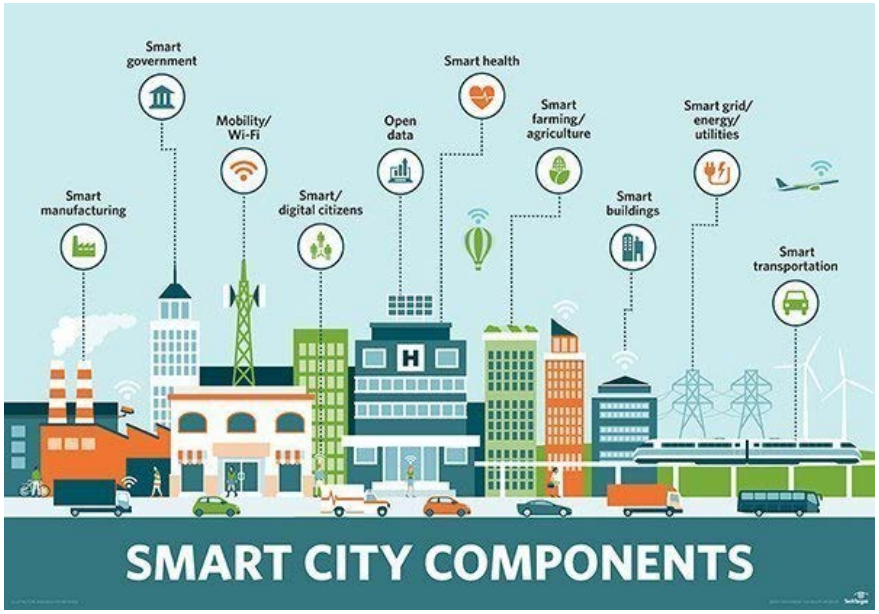
Presented at the NGOSCPS Workshop, CPSWeek 2019, Montreal, Canada

April 15, 2019

Great Minds Think Alike !!

- Self praise for all the participants of this workshop ☺
  - Because several ideas presented in this workshop are synergistic and have many common goals but seem to stem from different reasons and/or different motivating scenarios
- Work we present in this presentation stem from our work in cloud computing for CPS/IoT

Motivation: Scale of CPS/IoT Systems

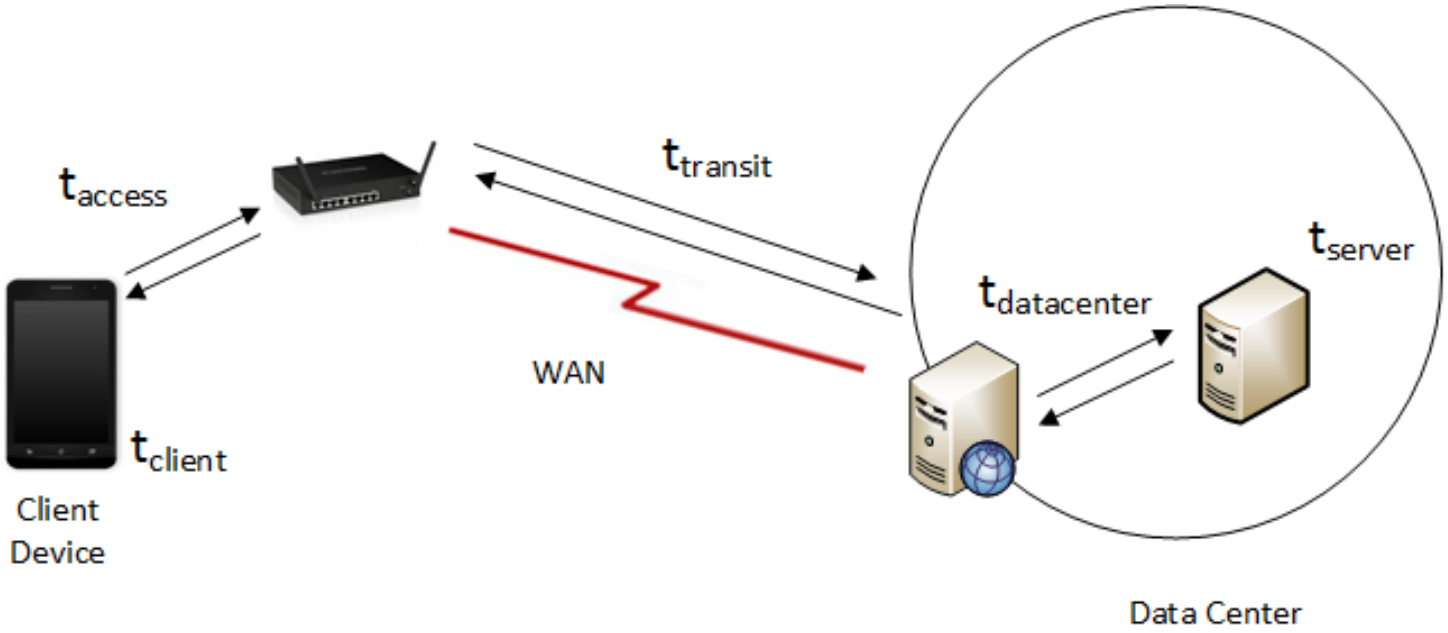


- Scale of CPS/IoT systems continue to grow
- Many interacting subsystems with their own QoS needs
- High degree of heterogeneity in resources, prog languages used to build systems, protocols used, etc
- Many interaction patterns (client-server, pub/sub, P2P, stream processing)
- Significant and differing levels of uncertainty in different parts of the CPS
- Increasing use of cloud resources for data-driven, compute-intensive needs

Cloud Latencies Hurtful for CPS/IoT

- End-to-end (round trip) latency for cloud-hosted IoT applications is computed as:

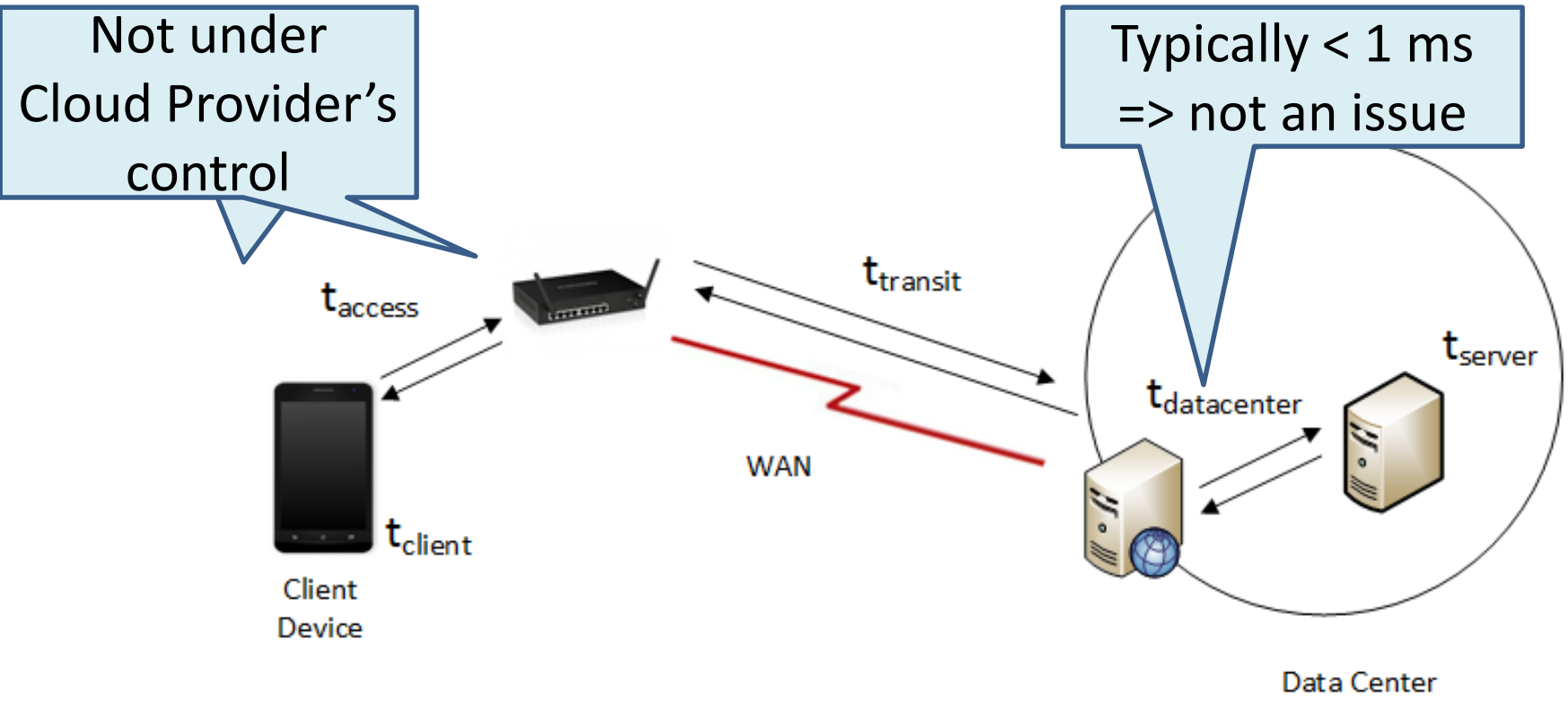
$$t_{total} = t_{client} + t_{access} + t_{transit} + t_{datacenter} + t_{server}$$



Cloud Latencies Hurtful for CPS/IoT

- End-to-end (round trip) latency for cloud-hosted IoT applications is computed as:

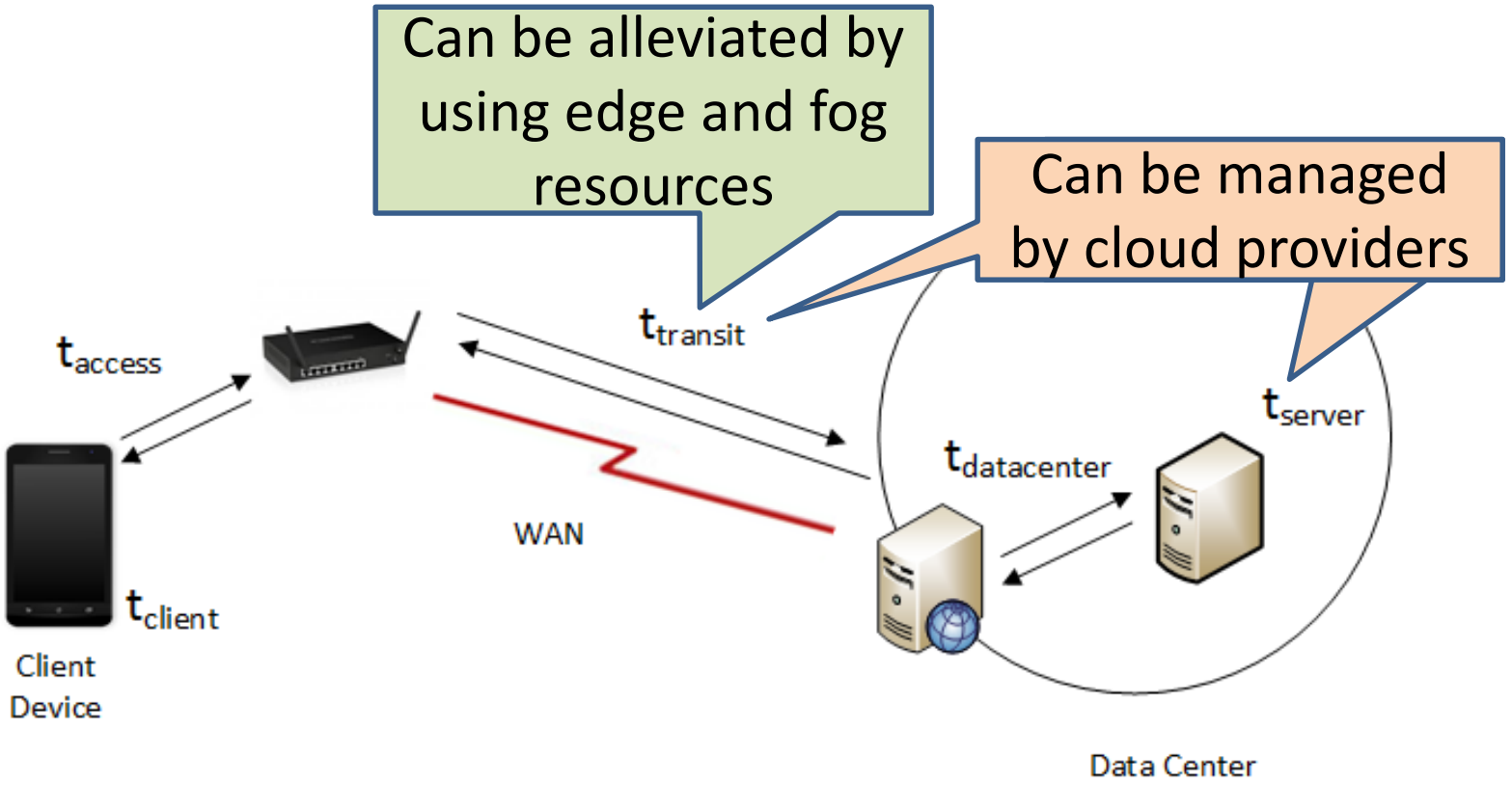
$$t_{total} = t_{client} + t_{access} + t_{transit} + t_{datacenter} + t_{server}$$



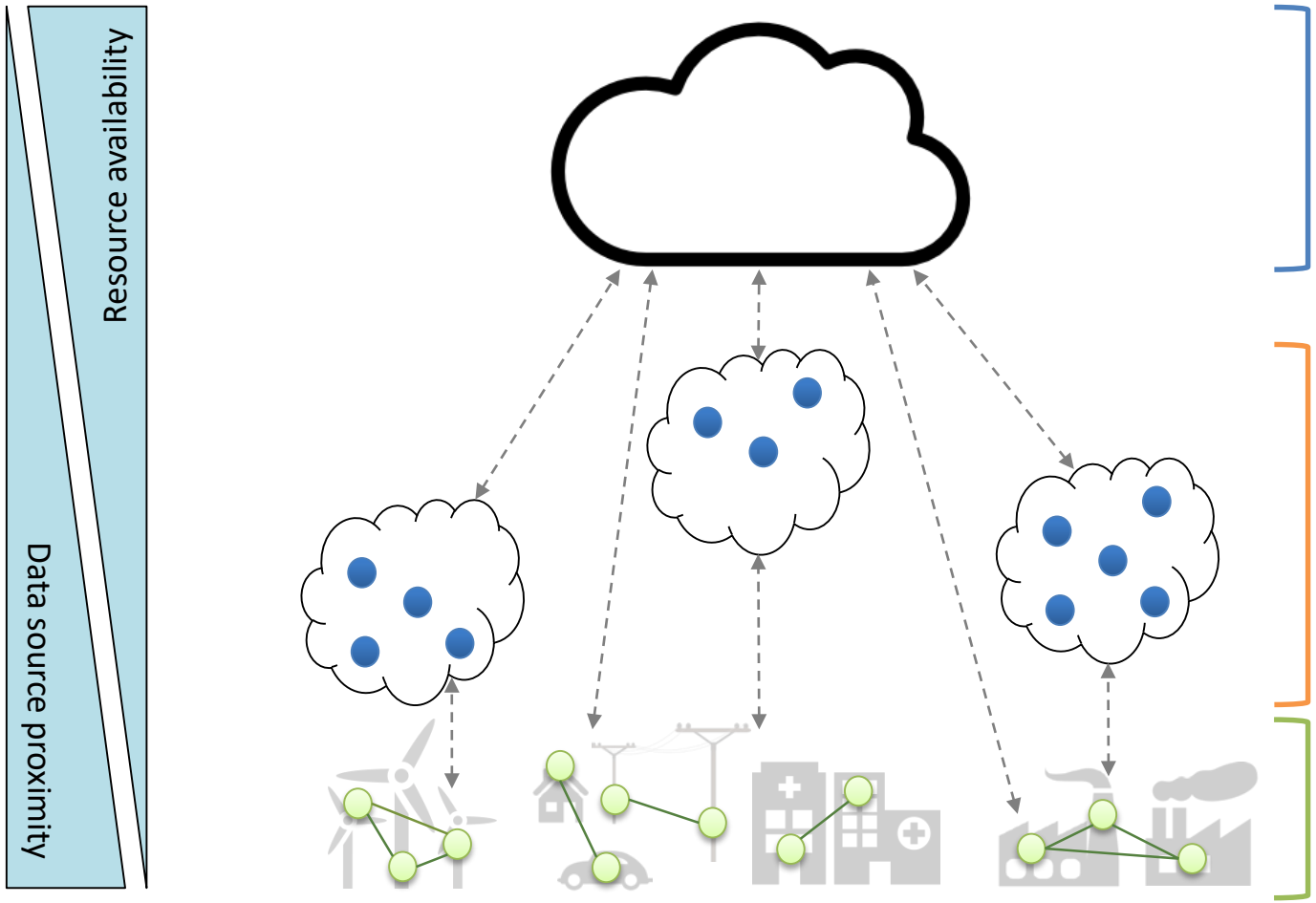
Cloud Latencies Hurtful for CPS/IoT

- End-to-end (round trip) latency for cloud-hosted IoT applications is computed as:

$$t_{total} = t_{client} + t_{access} + t_{transit} + t_{datacenter} + t_{server}$$



A 3-tiered Cloud Architecture for CPS/IoT



Challenges due to the 3-tier Model

- Different CPS/IoT subsystems leverage different tiers according to their QoS needs
  - Different elastic (vertical/horizontal) demands and autoscaling of resources needed for the different parts
  - Dynamic (re)consolidation of different subsystem tasks across the tiers
- Cannot assume same levels of network connectivity and available bandwidth
  - Network connectivity may fluctuate causing transient partitioning of the system at the edge/fog tiers
- Longer-term failures of system resources and disruptions
  - Sensor failures are common
- Performance monitoring and resource utilizations at all tiers required and a global picture is needed

Distributed Systems Issues Galore !!

- **CAP Theorem:** Consistency-Availability-Partitioning (when shared state is involved)
  - Different subsystems may make different trade-offs on whether to favor consistency or availability during partitioning
  - How to reconcile these differences between subsystems when they must interact with each other?
- **Coordination**, **Consensus**, **Distributed Snapshot**, **Discovery**, **Distributed & Dynamic Resource Management** and the list goes on …

Current State of the Operating System

- Totally unaware of its place and presence in the (large) distributed system
  - Existing OS typically deals with all the issues specific to only a single device that it manages
    - e.g., scheduling, concurrency, synchronization, socket APIs, memory mgmt., paging, file systems, etc
- No ability for self-instrumentation and data-driven model building
  - Limited ability to collect system metrics at fine grained level, provide data-driven performance model building, and understand and address performance interference issues due to contention for non partitionable hardware resources
  - Not much in terms of distributed learning for system-scale model development

Incurred Inherent/Accidental Complexities

- CPS/IoT system designers required to incorporate several 3rd party frameworks
  - ZooKeeper (coordination), Paxos/Raft (consensus), InfluxDB (time series db), collectd (metrics collection), RabbitMQ/ZMQ/Kafka/DDS (messaging), Spark/Flink (stream processing), and many more
- Monumental effort expended in integrating and interoperating these solutions
- Each framework (due to its limitations) makes the overall system brittle and unmanageable

Proposal for Software-defined NGOSCPS

- Be aware of the distributed nature => Native support for fundamental distributed systems semantics and features
- Cannot let OS footprint grow arbitrarily => fine-grained composition of required features (e.g., via components or microservices)
- Capability for adaptive and dynamic replacement and refinement of composed features
- Ability to self-instrument, aware of new hardware capabilities, and provision for model learning
- Verifiable composition of NGOSCPS
- All of this should be software-defined

Why & What about Software-defined?

- Software-defined Infrastructure is a generalization of the Software-defined Networking concept
- Clean separation between the control plane (which is programmable and where all the intelligence lies) and the data plane (which does the actual work)
- SD-NGOSPCS will therefore comprise the programmable control plane where the feature composition and adaptive reconfiguration logic can reside
- Control plane can monitor the behavior of data plane and make adaptations
- Control planes of device-specific OS instances can coordinate to form a *distributed-aware* NGOSCPS

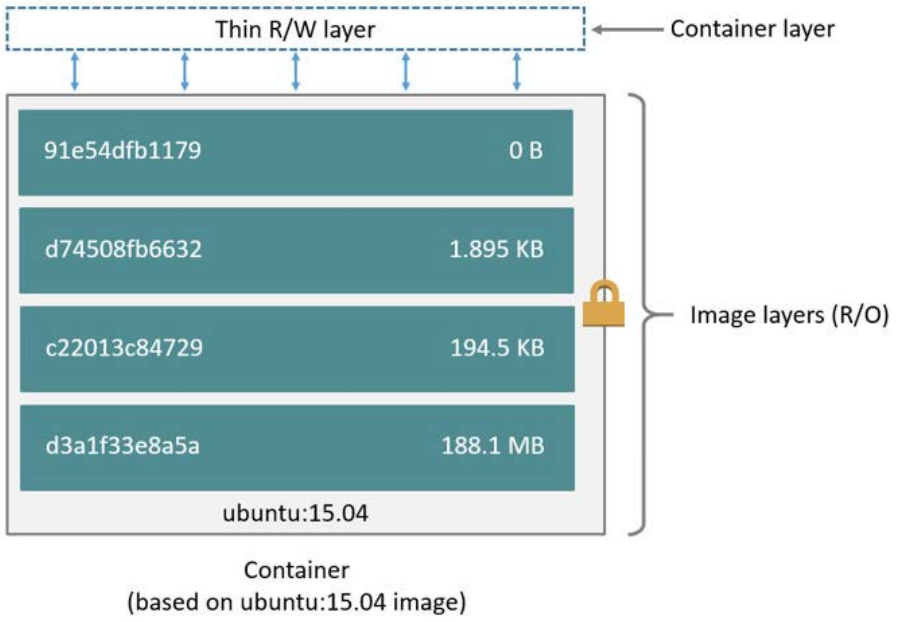
Approaches to Composition & Deployment

```
#
# Super simple example of a Dockerfile
#
FROM ubuntu:latest
MAINTAINER Andrew Odewahn "odewahn@oreilly.com"

RUN apt-get update
RUN apt-get install -y python python-pip wget
RUN pip install Flask

ADD hello.py /home/hello.py

WORKDIR /home
```



- Declarative approach similar to how a Docker image is described and built
- NGOSCPS images can be maintained similar to Docker images, which use OverlayFS
- In some respect similar to microkernels and how its features are composed
- DevOps tools like Ansible/Chef can be used to managed distributed deployment/customization

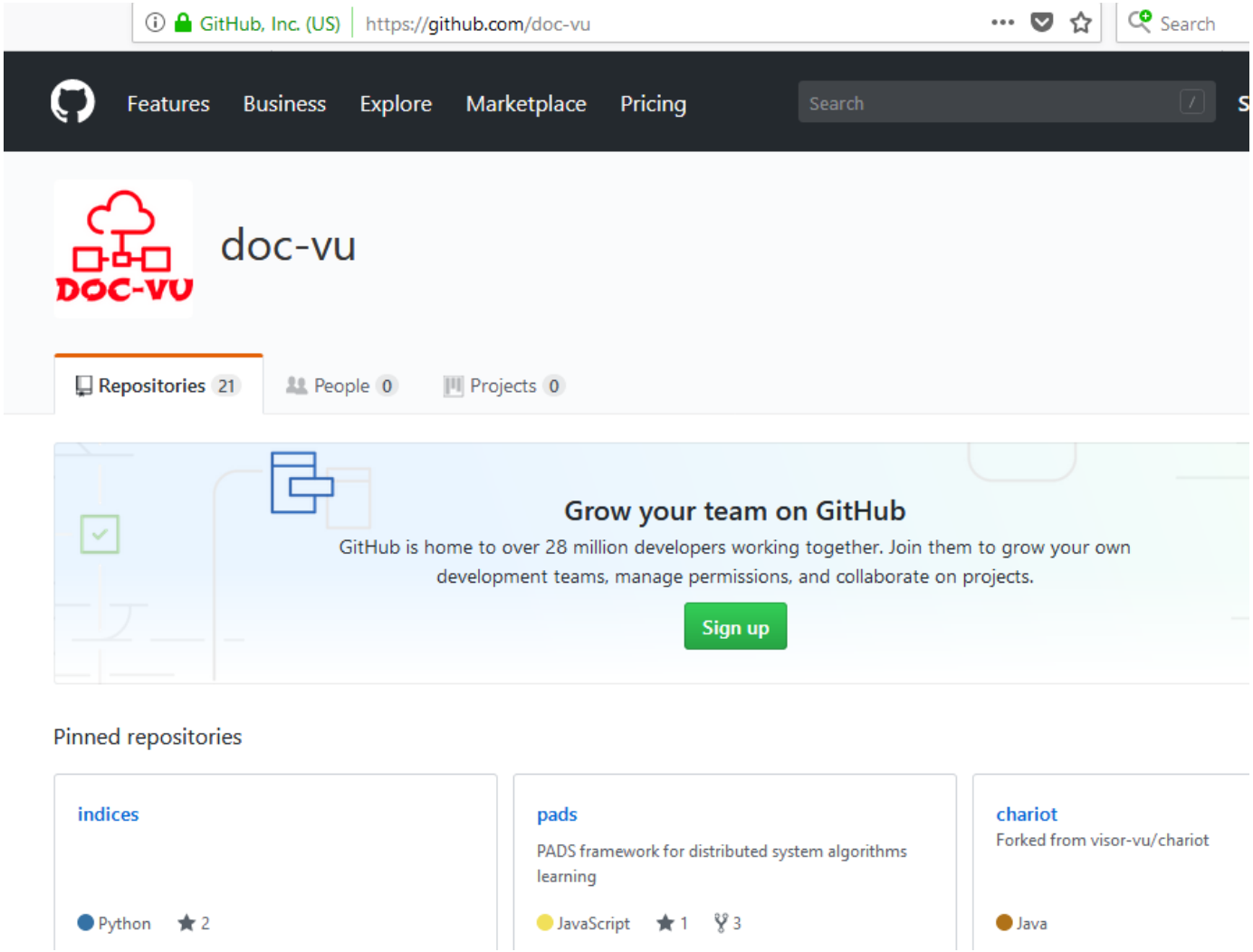
Synergies with Workshop Participants' Ideas (1/2)

- Branden's paper calls for a OS that is aware of distributed computing – exactly similar to our argument
- Like Bjorn's TOROS idea that can catch non compatible compositions at design-time, however, our approach is not opinionated (however, we can document proven patterns)
- Supports Arne's ideas of supporting heterogeneous applications and good isolation
- Fully synergistic with Nils' and Gabriel's pluggable, componentized design ideas
- Can benefit from Hudson's language-based type safety for network security
  - Maybe the control plane can be realized using this approach
- Supports Hendrik's idea that the OS needs to be aware of the full capability of the hardware and map the application's needs onto the best features of the hardware
  - Maybe use this to migrate tasks to the most capable hardware

Synergies with Workshop Participants' Ideas (2/2)

- Fully synergistic with Sandeep's idea of distributed coordination and need to see how to leverage the time/location ideas he has put forth
  - Maybe extend the distributed systems algorithms with these concepts and making them first class citizens of NGOSCPS
- Need to understand how Dio's idea of distributed temporal physical clocks and David's physical dynamics can be complemented with the distributed systems ideas we proposed (after all we are focused on cyber-physical)
  - Need to redesign many of the algorithms to consider physical clocks and dynamics
- The 3-tier research I presented could benefit from Andrea's SR-IoT (network function chain)
- We will benefit from the security ideas proposed in Bryan's paper by making security as a first class consideration in composition of needed features in the OS
- Scheduling the composed OS' functionality may benefit from LP-FPS ideas proposed by Reinder

Our Projects: https://github.com/doc-vu



THANK YOU

QUESTIONS?